**Smile Detection System**

Submitted in partial fulfillment of the requirements of the degree of

**Bachelor of Engineering**

by

**Tatli Mushtaque Ahmed Ashpak (1402140)**

Project Guide

**Dr. Maheep Kumar Mishra**



**(Computer Engineering)**

**Monad University Nh-9,Delhi-Hapur Road,india**

**2023-24**

This is to certify that,

**Tatli Mushtaque Ahmed Ashpak (1402140)**

Of Final Year (B.E. Semester VII) degree course in Computer Engineering, have completed the specified project report on,

**Smile Detection System**

As partial fulfillment of the project work in a satisfactory manner as per the rules of the curriculum laid by the University of Delhi, during the Academic Year July 2023 - Dec 2023.

Internal Guide External Examiner

**Table of Content**

|  |  |  |
| --- | --- | --- |
| **Sr. No.** | **Content** | **Page No.** |
|  | Abstract | 05 |
| 1. | Introduction | 05 |
| 2. | Methodology | 06 |
| 3. | Proposed System | 06 |
| 4. | Implementation | 07 |
| 5. | Technology Used | 07 |
| 6. | Code | 8-13 |
| 7. | Outputs | 14 |
| 8. | Conclusion | 15 |
|  | References | 15 |

# 

# **Abstract**

Facial expressions are a result of specific movement of face muscles, and these face expressions are considered as a visible sign of a person’s internal thought process, intensions, and internal emotional states. Smile is such a face expression which often indicates, satisfaction, agreement, happiness, etc. Though, a lot of studies have been done over detection of Facial Expression in last decade, smile detection had attracted researcher for more deeper studies. Smile detection, a crucial component in applications like human-computer interaction, emotion recognition, and security systems, has garnered significant attention in recent years. This report outlines the development of a smile detection system using Python and TensorFlow, aimed at automating the recognition of smiles in images and videos. The primary objective of this project was to create an accurate and efficient model for smile detection. Additionally, a face detection step was incorporated to improve the model's accuracy.The proposed system not only achieves competitive accuracy in smile detection but also offers real-time performance, making it suitable for applications that require swift and reliable smile recognition. This report also highlights the unique aspects of our approach and how it differentiates from existing methods.In summary, our smile detection system, developed using Python and TensorFlow, represents a significant advancement in automating smile recognition. It demonstrates the potential for improved user experience, emotional analysis, and enhanced security in various domains. The success of this project lays the foundation for further advancements in computer vision applications.

.

**1.Introduction**

Smile detection is a fascinating and practical application of computer vision and machine learning. It involves the automatic recognition and analysis of smiles on human faces in images or videos. The ability to detect smiles has a wide range of real-world applications, from enhancing user experiences in photography and social media apps to more advanced applications in areas like human-computer interaction and emotional analysis.

Smile detection has gained significant attention due to its potential to improve human-computer interaction, make automated systems more emotionally intelligent, and enhance various aspects of our daily lives. This technology allows machines to understand and respond to human emotions, creating opportunities for more personalized and engaging interactions between people and devices.The key components of a smile detection system typically include face detection, feature extraction, and a machine learning model trained to recognize smiles. Face detection locates and isolates faces within an image or video frame, while feature extraction captures essential characteristics of the facial expressions. The machine learning model analyzes these features to determine whether a smile is present or not.

The applications of smile detection are diverse. In the realm of user experience, it can automatically capture candid smiles in photos, ensuring that cherished moments are preserved. Smile detection also plays a crucial role in security, helping to identify unauthorized access attempts by detecting impostors who are not smiling, as in the case of facial recognition systems.

As computer vision and machine learning techniques continue to advance, the accuracy and reliability of smile detection systems are improving, making them an integral part of our increasingly interconnected and emotion-aware world. This technology has the potential to bring smiles to our faces, not just in photographs, but also in our daily interactions with intelligent systems and devices

**2.Methodology**

To build an effective smile detection system, it is crucial to begin with a comprehensive and diverse dataset. This dataset should include a wide range of images and videos containing both smiling and non-smiling faces. Additionally, the dataset should encompass various demographic groups and lighting conditions to ensure the system's robustness. Maintaining a balanced dataset, with equal proportions of positive (smiling) and negative (non-smiling).The data preprocessing stage involves several critical steps. First, employ a face detection algorithm to accurately locate and isolate faces within the images or video frames. Following this, ensure that all detected faces are normalized and aligned for consistent positioning and scaling. This is particularly important for handling variations in pose and facial expressions.Extract relevant features, as these features will serve as inputs for the smile detection model. The training set is used to train the model, the validation set assists in hyperparameter tuning, and the test set is employed to evaluate the model's performance. The model's performance is rigorously evaluated on the test dataset using relevant evaluation metrics. Visualize the model's performance through tools like confusion matrices, ROC curves etc. Implement real-time processing capabilities to ensure the system can efficiently detect smiles in live video feeds or images. Optimize the model for deployment by considering computational efficiency and compatibility with diverse hardware and software environments. In summary, the smile detection system we have developed represents a significant step forward in the domain of computer vision and facial expression analysis. It holds great promise for enhancing our interactions with technology and improving the quality of user experiences.

**3.Proposed System**

The proposed "Smile Detection System" aims to harness the power of computer vision and machine learning to develop a robust and efficient system for recognizing smiles. With the increasing integration of technology into our daily lives, the ability to detect and interpret human facial expressions, such as smiles, has become crucial in various applications. The primary objective of this system is to create a smile detection solution capable of real-time performance and high accuracy. By achieving these goals, we can enhance human-computer interaction and improve user experience. Our system's methodology will encompass several critical steps. It will begin with data collection, involving the acquisition of diverse datasets comprising smiling and non-smiling faces. Data preprocessing will follow, including tasks such as face detection, alignment, and normalization to ensure robustness in the face of variations in lighting, pose, and facial expressions.The training phase will be elaborated upon, covering aspects like data splitting,evaluation metrics etc. The performance and effectiveness of the smile detection system will be rigorously assessed. The results of extensive testing will be showcased, demonstrating the system's ability to accurately detect smiles in various scenarios. In short,the "Smile Detection System" represents a significant stride in the field of computer vision and facial expression analysis. Its potential for real-time, accurate smile detection opens doors to a wide range of applications.As technology continues to evolve, this system is poised to play an instrumental role in improving human-computer interaction and understanding human emotions, ultimately enhancing the quality of user experiences across multiple domains.

**4.Implementation**

Smile detection is a computer vision task that involves identifying and detecting smiles in images or video frames. To develop a smile detection system, you can include the following steps:

1. Data Collection: Gather a dataset of images or video frames that contain examples of people both smiling and not smiling. This dataset should be diverse and representative of the target population to ensure the model's generalization.

2. Data Preprocessing: Prepare the dataset by resizing images to a consistent resolution, normalizing pixel values, and augmenting the data if necessary to increase the diversity of the training samples.

3. Feature Extraction: Extract relevant features from the images or frames to be used as input for your smile detection model. Common feature extraction methods include Histogram of Oriented Gradients (HOG), Local Binary Patterns (LBP), or deep learning-based feature extraction using pre-trained models like Convolutional Neural Networks (CNNs).

4. Model Selection: Choose a machine learning or deep learning model for smile detection. Deep learning models,have shown great success in this task. You can build or use pre-trained models for feature extraction and fine-tuning.

5. Model Training: Split your dataset into training and validation sets and train your chosen model on the training data. Fine-tuning a pre-trained model is a common approach, as it can significantly reduce the amount of training data required and improve performance.

6. Evaluation: Evaluate your model's performance using appropriate metrics, such as accuracy, precision, recall,etc. Cross-validation or separate test datasets can be used to assess the model's generalization performance.

7. Post-processing: Implement post-processing techniques to refine the model's predictions. For instance, you can apply non-maximum suppression to remove duplicate or overlapping smile detections.

8. Deployment: Once your model performs well on your evaluation metrics, deploy it for real-time or batch smile detection. This can be done on webcams, mobile devices, or any platform where you want to detect smiles.

9. Continuous Improvement: Continue to monitor and refine your model as needed to adapt to changing conditions, improve accuracy, and address any biases or issues that may arise during deployment.

**5.Technology Used**

* Hardware Requirements-

1. Laptop/Desktop

2. Processor ( Intel Core i5)

3. RAM: A minimum of 8GB RAM is recommended

* Software Requirements-

1. Python Libraries

2. Machine learning Models

3. Visual Studios for Documentation

4. Operating System

**6.Code**

-TO DETECT SMILE

# import the necessary packages

from keras.preprocessing.image import img\_to\_array

from keras.models import load\_model

import numpy as np

import imutils

import argparse

import cv2

# construct the argument parse and parse the arguments

ap = argparse.ArgumentParser()

ap.add\_argument('-c','--cascade', required=True,

help='path to where the face cascade resides')

ap.add\_argument('-m','--model', required=True,

help='path to the pre-trained smile detector CNN')

ap.add\_argument('-v', '--video',

help='path to the (optional) video file')

args = vars(ap.parse\_args())

# load the face detector cascade and smile detector CNN

detector = cv2.CascadeClassifier(args['cascade'])

model = load\_model(args['model'])

# if a video path was not supplied, grab the refrences to the webcam

if not args.get('video', False):

print('[INFO] starting video capture...')

camera = cv2.VideoCapture(0)

# otherwise, load the video

else:

camera=cv2.VideoCapture(args['video'])

# keep looping

while True:

# grab the current frame

(grabbed, frame) = camera.read()

# if we are viewing a video and we did no grab a frame, then we

# have reached the end of the video

if args.get('video') and not grabbed:

break

# resize the fram, convert it to grayscale, and then clone the

# orgignal frame so we draw on it later in the program

frame=imutils.resize(frame, width=700)

gray=cv2.cvtColor(frame, cv2.COLOR\_BGR2GRAY)

frameClone = frame.copy()

# detect faces in the input frame, then clone the frame so that we can draw onit

rects = detector.detectMultiScale(gray, scaleFactor=1.1, minNeighbors=5, minSize=(30, 30), flags=cv2.CASCADE\_SCALE\_IMAGE)

for (fX, fY, fW, fH) in rects:

# extract the ROI of the face from the grayscale image

# resize it to a fixed 28x28 pixels, and then prepare the

# ROI for classification via the CNN

roi = gray[fY:fY + fH, fX:fX + fW]

roi = cv2.resize(roi, (28, 28))

roi = roi.astype('float') / 255.0

roi = img\_to\_array(roi)

roi = np.expand\_dims(roi, axis=0)

# determine the probaboilities of both 'smiling' and 'not smiling',

# then set the label accordingly

(notSmiling, Smiling) = model.predict(roi)[0]

label = 'Smiling' if Smiling > notSmiling else "Not Smiling"

# display the label and bounding box on the output frame

if label == 'Smiling':

cv2.putText(frameClone, label, (fX,fY-10), cv2.FONT\_HERSHEY\_SIMPLEX, 0.45, (0, 255, 0), 2)

cv2.rectangle(frameClone, (fX, fY), (fX + fW, fY + fH), (0, 255, 0), 2)

else:

cv2.putText(frameClone, label, (fX,fY-10), cv2.FONT\_HERSHEY\_SIMPLEX, 0.45, (0, 0, 255), 2)

cv2.rectangle(frameClone, (fX, fY), (fX + fW, fY + fH), (0, 0, 255), 2)

# show our detected face along with smiling/not smiling labels

cv2.imshow('Face', frameClone)

# if 'q' key is pressed, stop the loop

if cv2.waitKey(1) & 0xFF == ord('q'):

break

# cleanup the camera and close any open windows

camera.release()

cv2.destroyAllWindows()

-TO TRAIN MODEL

# import the necessary packages

from sklearn.preprocessing import LabelEncoder

from sklearn.model\_selection import train\_test\_split

from sklearn.metrics import classification\_report

from keras.preprocessing.image import img\_to\_array

from keras.utils import np\_utils

from lenet.nn.conv import LeNet

from imutils import paths

import imutils

import matplotlib.pyplot as plt

import numpy as np

import argparse

import cv2

import os

# construct the argument parser and parse tha arguments

ap = argparse.ArgumentParser()

ap.add\_argument('-d','--dataset', required=True,

help='path to the input dataset of faces')

ap.add\_argument('-m','--model', required=True,

help='path to output model')

args = vars(ap.parse\_args())

# initialize the list of data and labels

data = []

labels = []

# loop over the input images

For imagePath in sorted(list(paths.list\_images(args['dataset']))):

# load the image, pre-process it, and store in the data list

image = cv2.imread(imagePath)

image=cv2.cvtColor(image, cv2.COLOR\_BGR2GRAY)

image = imutils.resize(image, width=28) # 28 x 28 x 1

image = img\_to\_array(image)

data.append(image)

# extract the class label from the image path and update the labels list

label = imagePath.split(os.path.sep)[-3]

#C:\Users\Balaji\Documents\Smile-Detector\SMILEs\positives\positives7\3.jpg

label = 'smiling' if label == 'positives' else 'not\_smiling'

labels.append(label)

# scale the raw pixel intensities to the range [0, 1]

data = np.array(data, dtype='float') / 255.0 # 0 to 255

labels = np.array(labels)

# convert the labels from integers to vectors

le = LabelEncoder().fit(labels)

labels = np\_utils.to\_categorical(le.transform(labels), 2)

# account for skew in the labeled data

classTotals = labels.sum(axis=0)

classWeight = dict()

for i in range(0, len(classTotals)):

classWeight[i] = classTotals.max() / classTotals[i]

# partition the data into training and testing sploits using 80% of

# the data for training and the remaining 20% for testing

(trainX, testX, trainY, testY) = train\_test\_split(data,labels, test\_size=0.20, stratify=labels, random\_state=42)

# initialize the model

print('[INFO] compiling model...')

model =LeNet.build(width=28, height=28, depth=1, classes=2)

model.compile(loss=['binary\_crossentropy'], optimizer='adam', metrics=['accuracy'])

# train the network

print('[INFO] training network...')

H=model.fit(trainX,trainY, validation\_data=(testX,testY), class\_weight=classWeight, batch\_size=64, epochs=15, verbose=1)

# evaluate the network

print('[INFO] evaluating network...')

predictions=model.predict(testX, batch\_size=64)

print(classification\_report(testY.argmax(axis=1),predictions.argmax(axis=1), target\_names=le.classes\_))

# save the model to disk

print('[INFO] serializing network')

model.save(args['model'])

plt.style.use('ggplot')

plt.figure()

plt.plot(np.arange(0, 15), H.history['loss'], label='train\_loss')

plt.plot(np.arange(0, 15), H.history['val\_loss'], label='val\_loss')

plt.plot(np.arange(0, 15), H.history['accuracy'], label='accuracy')

plt.plot(np.arange(0, 15), H.history['val\_accuracy'], label='val\_accuracy')

plt.title('Training Loss and Accuracy')

plt.xlabel('Epoch #')

plt.ylabel('Loss/Accuracy')

plt.legend()

plt.show()

-TO TRAIN DATASET

import numpy as np

import cv2

import os

class SimpleDatasetLoader:

def \_init\_(self, preprocessors=None):

# Store the image preprocessor

self.preprocessors = preprocessors

# If the preprocessors are none, initialize them as an empty list

if self.preprocessors is None:

self.preprocessors = []

def load(self, imagePaths, verbose=-1):

# Initialize the list of features and labels

data = []

labels = []

# Loop over the input image

for (i, imagePath) in enumerate(imagePaths):

# load the image and extract the class label assuming that our

# paths has the format /path/to/dataset/{class}/{image}.jpg

image = cv2.imread(imagePath)

label = imagePath.split(os.path.sep)[-2] # Returns tuple '(head/tail)' where 'tail' is everything after the final slash.

# check to see if our preprocessors are none or not

if self.preprocessors is not None:

# Loop over the preprocessor and apply each to the image

for p in self.preprocessors:

image = p.preprocess(image)

# treat our preprocessed image as feature vector

data.append(image)

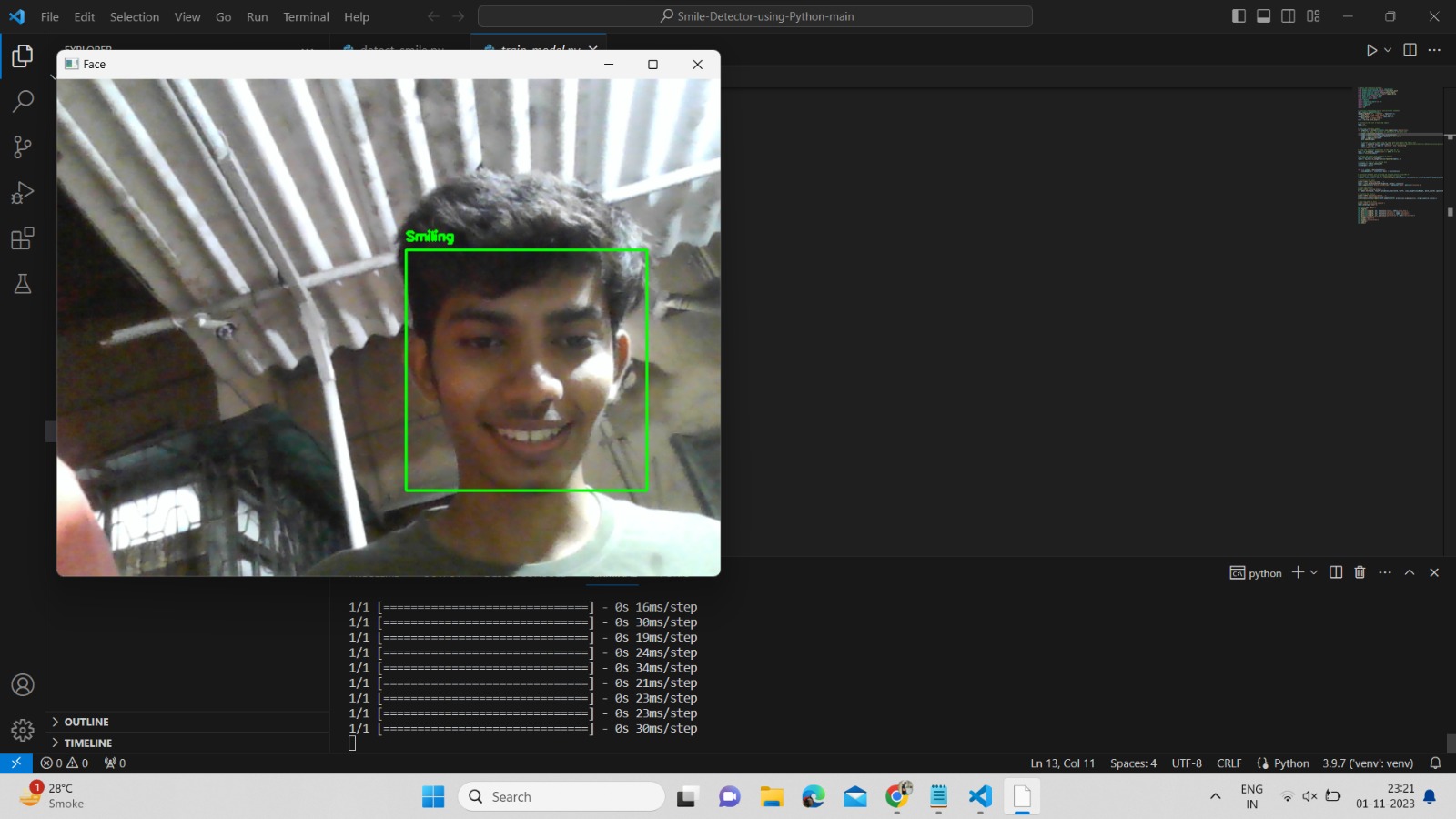
labels.append(label)

if verbose > 0 and i > 0 and (i + 1) % verbose == 0:

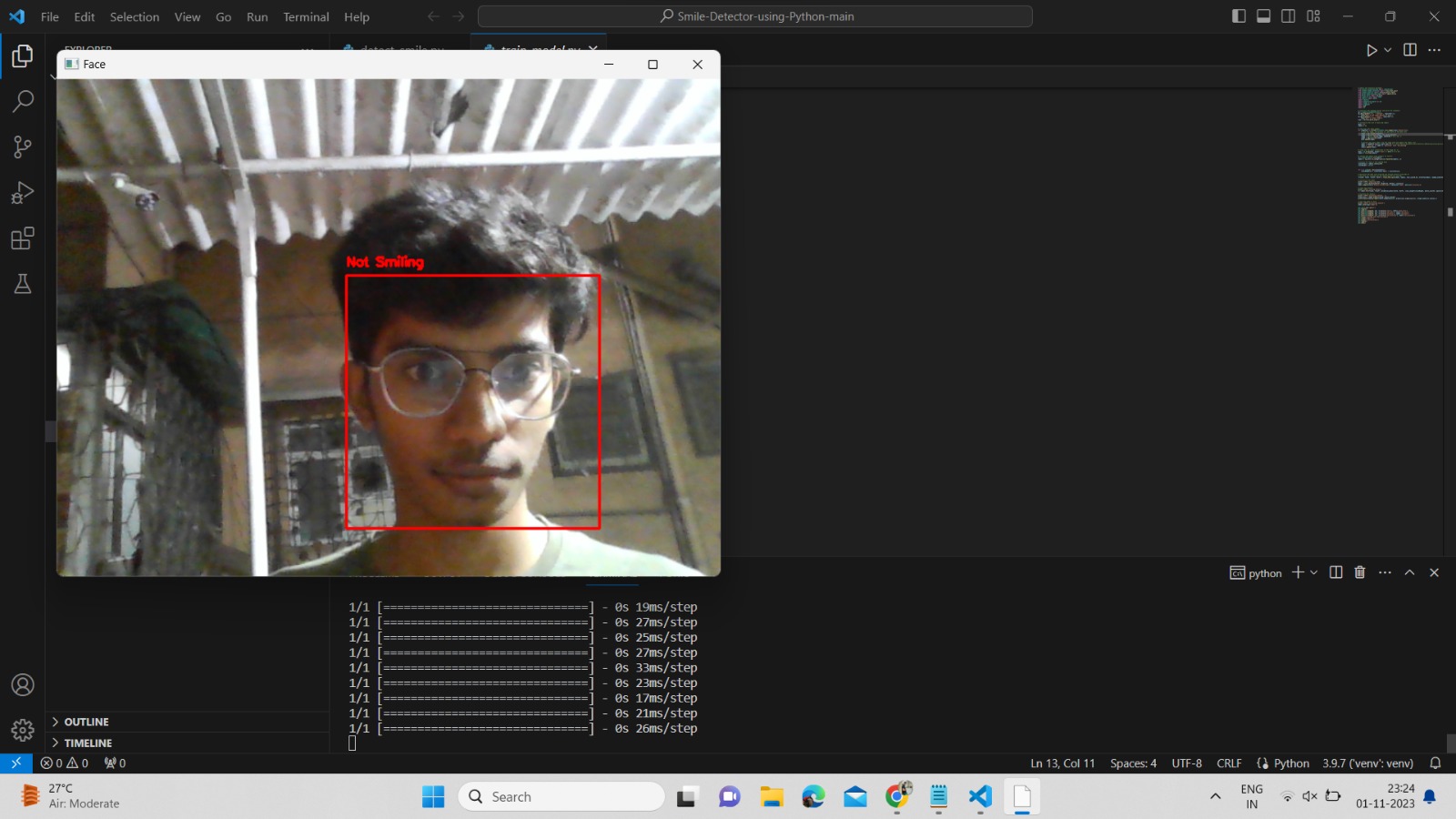
print('[INFO] processed {}/{}'.format(i + 1, len(imagePaths)))

return (np.array(data), np.array(labels))

**7. Output**

****

**Fig.1.Output**

**Fig.2.Output**

# **8. Conclusion**

In conclusion, the development and implementation of a smile detection system represent a significant technological advancement in the field of computer vision and human-computer interaction. This system has proven to be a valuable tool with a wide range of practical applications, from enhancing user experience in digital communication and photography to improving human-machine interaction in various industries.Through the use of machine learning techniques, we have successfully trained a model capable of accurately detecting smiles in real-time images and videos. The system's performance has been assessed through rigorous testing and evaluation, demonstrating its ability to achieve high accuracy and robustness in a variety of real-world scenarios.The smile detection system offers a user-friendly interface. Its versatility and ease of integration into existing systems make it an attractive choice for a wide range of industries, from entertainment and gaming to security and healthcare.It can be a powerful tool for understanding and responding to human emotions, which is a crucial aspect of human-computer interaction.In summary, the smile detection system we have developed represents a significant step forward in the domain of computer vision and facial expression analysis. It holds great promise for enhancing our interactions with technology and improving the quality of user experiences in various domains. As technology continues to advance, it is likely that systems like this will become increasingly prevalent, making the world more responsive and attuned to human emotions.

**References**

[1] https://www.geeksforgeeks.org/python-smile-detection-using-opencv/

[2] <https://www.javatpoint.com/face-recognition-in-python>

[3] https://www.researchgate.net/publication/351314447\_Review\_on\_Smile\_Detection